Streaming and Massively Parallel Algorithms for Edge Coloring

Soheil Behnezhad
University of Maryland
soheil@cs.umd.edu

Mahsa Derakhshan
University of Maryland
mahsa@cs.umd.edu

MohammadTaghi Hajiaghayi
University of Maryland
hajiagha@cs.umd.edu

Marina Knittel
University of Maryland
mknittel@cs.umd.edu

Hamed Saleh
University of Maryland
hamed@cs.umd.edu

Abstract

A valid *edge-coloring* of a graph is an assignment of "colors" to its edges such that no two incident edges receive the same color. The goal is to find a proper coloring that uses few colors. (Note that the maximum degree, Δ, is a trivial lower bound.) In this paper, we revisit this fundamental problem in two models of computation specific to massive graphs, the *Massively Parallel Computations* (MPC) model and the *Graph Streaming* model:

Massively Parallel Computation. We give a randomized MPC algorithm that with high probability returns a $\Delta + \widetilde{O}(\Delta^{3/4})$ edge coloring in $O(1)$ rounds using $O(n)$ space per machine and $O(m)$ total space. The space per machine can also be further improved to $n^{1-\Omega(1)}$ if $\Delta = n^{\Omega(1)}$. Our algorithm improves upon a previous result of Harvey *et al.* [SPAA 2018].

Graph Streaming. Since the output of edge-coloring is as large as its input, we consider a standard variant of the streaming model where the output is also reported in a streaming fashion. The main challenge is that the algorithm cannot "remember" all the reported edge colors, yet has to output a proper edge coloring using few colors.
We give a one-pass $\widetilde{O}(n)$-space streaming algorithm that always returns a valid coloring and uses 5.44Δ colors with high probability if the edges arrive in a random order. For adversarial order streams, we give another one-pass $\widetilde{O}(n)$-space algorithm that requires $O(\Delta^2)$ colors.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Massively Parallel Computation, Streaming, Edge Coloring

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.14

Acknowledgements Supported in part by Guggenheim Fellowship, NSF grants CCF:SPX 1822738, IIS:BIGDATA 1546108, DARPA grant SI3CMD, UMD Year of Data Science Program Grant, and Northrop Grumman Faculty Award.

1 Introduction

Given a graph $G(V, E)$, an edge coloring of G is an assignment of "colors" to the edges in E such that no two incident edges receive the same color. The goal is to find an edge coloring that uses few colors. Edge coloring is among the most fundamental graph problems and has been studied in various models of computation, especially in distributed and parallel settings.

Denoting the maximum degree in the graph by Δ, it is easy to see that Δ colors are necessary in any proper edge coloring. On the other hand, Vizing's celebrated theorem asserts that $\Delta + 1$ colors are always sufficient [39]. While determining whether a graph can be Δ colored is NP-hard, a $\Delta + 1$ coloring can be found in polynomial time [5, 21]. These algorithms are, however, highly sequential. As a result, in restricted settings, it is standard to consider more relaxed variants of the problem where more colors are allowed [2, 8, 20, 24, 25, 27, 29, 32, 34, 35, 36].

In this paper, we study edge coloring in large-scale graph settings. Specifically, we focus on the *Massively Parallel Computations* (MPC) model and the *Graph Streaming* model.

1.1 Massively Parallel Computation

The Model. The MPC model [10, 26, 31] is a popular abstraction of modern parallel frameworks such as MapReduce, Hadoop, Spark, etc. In this model, there are N machines, each with a space of S words[1] that all run in parallel. The input, which in our case is the edge-set of graph $G(V, E)$, is initially distributed among the machines arbitrarily. Afterwards, the system proceeds in synchronous rounds wherein the machines can perform any arbitrary local computation on their data and can also send messages to other machines. The messages are then delivered at the start of the next round so long as the total messages sent and received by each machine is $O(S)$ for local machine space S. The main parameters of interest are S and the round-complexity of the algorithm, i.e., the number of rounds it takes until the algorithm stops. Furthermore, the total available space over all machines should ideally be linear in the input size, i.e., $S \cdot N = O(|E|)$.

Related Work in MPC. We have seen a plethora of results on graph problems ever since the formalization of MPC. The studied problems include matching and vertex cover [1, 6, 14, 17, 22, 33, 12, 15], maximal independent set [22, 28, 12, 15], vertex coloring [7, 16, 28, 37, 38], as well as graph connectivity and related problems [3, 4, 13, 30, 9]. (This is by no means a complete list of the prior works.)

We have a good understanding of the complexity of vertex coloring in the MPC model, especially if the local space is near linear in n: Assadi et al. [7] gave a remarkable algorithm that using $\widetilde{O}(n)$ space per machine, finds a $(\Delta + 1)$ vertex coloring in a constant number of rounds. The algorithm is based on a sparsification idea that reduces the number of edges from m to $O(n \log^2 n)$. But this algorithm alone cannot be used for coloring the edges, even if we consider the more relaxed $(2\Delta - 1)$ edge coloring problem which is equivalent to $(\Delta + 1)$ vertex coloring on the line graph. The reason is that the line-graph has $O(m)$ vertices where here m is the number of edges in the original graph. Therefore even after the sparisification step, we have $\widetilde{O}(m)$ vertices in the graph which is much larger than the local space available in the machines.

[1] Throughout the paper, the stated space bounds are in the number of words that each denotes $O(\log n)$ bits.

Not much work has been done on the edge coloring problem in the MPC model. The only exception is the algorithm of Harvey *et al.* [28] which roughly works by random partitioning the *edges*, and then coloring each partition in a different machine using a sequential $(\Delta+1)$ edge coloring algorithm. The choice of the number of partitions leads to a trade-off between the number of colors used and the space per machine required. The main shortcoming of this idea, however, is that if one desires a $\Delta + \widetilde{O}(\Delta^{1-\Omega(1)})$ edge coloring, then a strongly super linear local space of $n\Delta^{\Omega(1)}$ is required.

Our main MPC result is the following algorithm which uses a more efficient partitioning. The key difference is that we use a *vertex* partitioning as opposed to the algorithm of Harvey *et al.* which partitions the edges.

▷ Result 1 (Theorem 1). There exists an MPC algorithm that using $O(n)$ space per machine and $O(m)$ total space, returns a $\Delta + \widetilde{O}(\Delta^{3/4})$ edge coloring in $O(1)$ rounds.

The algorithm exhibits a tradeoff between the space and the number of colors (see Theorem 1) and can be made more space-efficient as the maximum degree gets larger. For instance, if $\Delta > n^{\epsilon}$ for any constant $\epsilon > 0$, it requires a strictly sublinear space of $n^{1-\Omega(1)}$ to return a $\Delta + o(\Delta)$ edge coloring in $O(1)$ rounds. This is somewhat surprising since all previous non-trivial algorithms in the strictly sublinear regime of MPC require $\omega(1)$ rounds.

Our algorithm can also be implemented in $O(1)$ rounds of *Congested Clique*, leading to a $\Delta + \widetilde{O}(\Delta^{3/4})$ edge coloring there. Prior to our work, no sublogarithmic round Congested Clique algorithm was known even for $(2\Delta - 1)$ edge coloring.

1.2 Streaming

The Model. In the standard graph streaming model, the edges of a graph arrive one by one and the algorithm has a space that is much smaller than the total number of edges. A particularly important choice of space is $\widetilde{O}(n)$—which is also known as the *semi-streaming* model [19]—so that the algorithm has enough space to store the vertices but not the edges. For edge coloring, the output is as large as the input, thus, we cannot hope to be able to store the output and report it in bulk at the end. For this, we consider a standard twist on the streaming model where the output is also reported in a streaming fashion. This model is referred to in the literature as the "W-streaming" model [18, 23]. We particularly focus on one-pass algorithms.

Designing one-pass W-streaming algorithms is particularly challenging since the algorithm cannot "remember" all the choices made so far (e.g., the reported edge colors). Therefore, even the sequential greedy algorithm for $(2\Delta - 1)$ edge coloring, which iterates over the edges in an arbitrary order an assigns an available to each color upon visiting it, cannot be implemented since we are not aware of the colors used incident to an edge.

Our first result is to show that a natural algorithm w.h.p.[2] provides an $O(\Delta)$ edge coloring if the edges arrive in a random-order.

▷ Result 2 (Theorem 9). If the edges arrive in a random-order, there is a one-pass $\widetilde{O}(n)$ space W-streaming edge coloring algorithm that always returns a valid edge coloring and w.h.p. uses $(2e + o(1))\Delta \approx 5.44\Delta$ colors.

[2] Throughout, we use "w.h.p." to abbreviate "with high probability" implying probability at least $1 - 1/\operatorname{poly}(n)$.

If the edges arrive in an arbitrary order, we give another algorithm that requires more colors.

▷ Result 3 (Theorem 10). For any arbitrary arrival of edges, there is a one-pass $\widetilde{O}(n)$ space W-streaming edge coloring algorithm that succeeds w.h.p. and uses $O(\Delta^2)$ colors.

These are, to our knowledge, the first streaming algorithms for edge coloring.

2 The MPC Algorithm

In this section, we consider the edge coloring problem in the MPC model. Our main result in this section is an algorithm that achieves the following:

▶ **Theorem 1.** *For any parameter k (possibly dependent on Δ) such that $n/k \gg \log n$, there exists an* MPC *algorithm with $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ space per machine and $O(m)$ total space that w.h.p. returns a $\Delta + O(\sqrt{k\Delta \log n})$ edge coloring in $O(1)$ rounds.*

By setting $k = \sqrt{\Delta} + \log n$, the space required per machine will be $O(n)$ and the number of colors would be $\Delta + \widetilde{O}(\Delta^{3/4})$. Using a reduction from [11], this also leads to an $O(1)$ round Congested Clique algorithm using the same number of colors.

▶ **Corollary 2.** *There exists a randomized* MPC *algorithm with $O(n)$ local space, as well as a Congested Clique algorithm, that both w.h.p. find a $\Delta + \widetilde{O}(\Delta^{3/4})$ edge coloring in $O(1)$ rounds.*

Moreover, assuming that $\Delta = n^{\Omega(1)}$, by setting $k = \Delta^{0.5+\epsilon}$ for a small enough constant $\epsilon \in (0, 1)$, we get the following $O(1)$ round algorithm which requires $n^{1-\Omega(1)}$ machine space, which is notably strictly sublinear in n:

▶ **Corollary 3.** *If $\Delta = n^{\Omega(1)}$, there exists a randomized* MPC *algorithm with $O(n/\Delta^{2\epsilon}) = n^{1-\Omega(1)}$ space per machine and $O(m)$ total space that w.h.p. returns a $\Delta + \widetilde{O}(\Delta^{0.75+\epsilon/2})$ edge coloring in $O(1)$ rounds.*

The Idea Behind the Algorithm. The first step in the algorithm is a random partitioning of the *vertex set* into k groups, $V_1, \ldots, V_k$. We then introduce one subgraph for each vertex subset, called $G_1, \ldots, G_k$, and one subgraph for every pair of groups which we denote as $G_{1,2}, \ldots, G_{1,k}, \ldots, G_{k-1,k}$. Any such G_i is simply the induced subgraph of G on V_i. Moreover, any such $G_{i,j}$ is the subgraph on vertices $V_i \cup V_j$, with edges with one point in V_i and the other in V_j.

The general idea is to assign different *palettes*, i.e., subsets of colors, to different subgraphs so that the palettes assigned to any two neighboring subgraphs (i.e., those that share a vertex) are completely disjoint. A key insight to prevent this from blowing up the number of colors, is that since any two edges from $G_{i,j}$ and $G_{i',j'}$ with $i \neq i'$ and $j \neq j'$ cannot share endpoints by definition, it is safe to use the same color palette for them.

To assign these color palettes, we consider a complete k-vertex graph with each vertex v_i in it corresponding to partition V_i and each edge (v_i, v_j) in it corresponding to the subgraph $G_{i,j}$. We then find a k edge coloring of this complete graph, which exists by Vizing's theorem since maximum degree in it is $k-1$. This edge coloring can actually be constructed extremely efficiently using merely the edges' endpoint IDs. Thereafter, we map each of these k colors to a color palette. By carefully choosing k and the number of colors in each palette, we ensure that: (1) The total number of colors required is close to Δ. (2) Each subgraph $G_{i,j}$ can be properly edge-colored with those colors in its palette. (3) Each subgraph fits the memory of a single machine so that we can put it in whole there and run the sequential edge coloring algorithm on it.

Algorithm 1: An MPC algorithm for edge coloring.

```
Parameter: $k$.;
Output: An edge coloring of a given graph $G = (V, E)$ with maximum degree $\Delta$
        using $\Psi := \Delta + d\sqrt{k\Delta \log n}$ colors for some large enough constant $d$.
Independently and u.a.r. partition $V$ into $k$ subsets $V_1, \dots, V_k$.;
For every $i \in [k]$, let $G_i$ be the induced subgraph of $G$ on $V_i$.;
For every $i, j \in [k]$ with $i \neq j$, let $G_{i,j}$ be the subgraph of $G$ including an edge $e \in E$
  iff one end-point of $e$ is in $V_i$ and the other is in $V_j$.;
Partition $[\Psi]$ into $k + 1$ disjoint subsets $C_1, \dots, C_k, C'$, which we call color palettes, in
  an arbitrarily way such that each palette has exactly $\frac{\Psi}{k+1}$ colors.;
for each graph $G_i$ in parallel do
    Color $G_i$ sequentially in a single machine with palette $C'$.;
end
// In what follows, we implicitly construct a $k$ edge coloring of a complete $k$-vertex
   graph $K_k$ and assign palette $C_\alpha$ to subgraph $G_{i,j}$ where $\alpha$ is the color of edge
   $(i, j)$ in $K_k$.
for each graph $G_{i,j}$ in parallel do
    Color $G_{i,j}$ sequentially in a machine with palette $C_\alpha$ where
      $\alpha = ((i + j) \bmod k) + 1$.;
end
```

The algorithm outlined above is formalized as Algorithm 1. We start by proving certain bounds on subgraphs' size and degrees.

▷ Claim 4. W.h.p., every subgraph of type G_i or $G_{i,j}$ has maximum degree $\frac{\Delta}{k} + O(\sqrt{\Delta \log \frac{n}{k}})$ and has at most $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ edges.

Proof. Let us start with bounding the degree of an arbitrary vertex $v \in V_i$ in subgraph G_i. The degree of vertex v in G_i is precisely the number of its neighbors that are assigned to partition V_i. Since there are k partitions, the expected degree of v in G_i is $\deg_G(v)/k \leq \Delta/k$. Furthermore, since the assignment of vertices to the partitions is done independently and uniformly at random, by a simple application of Chernoff bound, v's degree in G_i should be highly concentrated around its mean. Namely, with probability at least $1 - n^{-2}$, it holds that $\deg_{G_i}(v) \leq \frac{\Delta}{k} + O(\sqrt{\Delta \log n/k})$. Now, a union bound over the n vertices in the graph, proves that the degree of all vertices in their partitions should be at most $\frac{\Delta}{k} + O(\sqrt{\Delta \log n/k})$ with probability $1 - 1/n$.

Bounding vertex degrees in subgraphs of type $G_{i,j}$ also follows from essentially the same argument. The only difference is that we have to union bound over $n \cdot k$ choices, as we would like to bound the degree of any vertex v with say $v \in V_i$ in k subgraphs $G_{i,1}, \dots, G_{i,k}$. Nonetheless, since $k \leq n$, there are still $\text{poly}(n)$ many choices to union bound over. Thus, by changing the constants in the lower terms of the concentration bound, we can achieve the same high probability result.

Finally, we focus on the number of edges in each of the subgraphs. Each partition V_i has n/k vertices in expectation since the n vertices are partitioned into k groups independently and uniformly at random. A simple application of Chernoff and union bounds, implies that the number of vertices in each partition V_i is at most $O(\frac{n}{k})$ w.h.p., so long as $n/k \gg \log n$, which is the case. Since the number of edges in each partition is less than the number of vertices times max degree, combined with the aforementioned bounds on the max degree, we can bound the number of edges in G_i and $G_{i,j}$ for any i and j by

$$O\Big(\frac{n}{k}\Big) \cdot O\left(\frac{\Delta}{k} + \sqrt{\frac{\Delta}{k}\log n}\right) = O\left(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\frac{\Delta}{k}\log n}\right),$$

which is the claimed bound. ◀

Next, observe that we use palettes $C_1, \ldots, C_{k+1}, C'$, each of size $\frac{\Psi}{k+1}$ to color the subgraphs. We need to argue that the maximum degree in each subgraph is at most $\frac{\Psi}{k+1} - 1$ to be able to argue that using Vizing's theorem in one machine, we can color any of the subgraphs with the assigned palettes. This can indeed be easily guaranteed if the constant d is large enough:

▶ **Observation 5.** *If constant d in Algorithm 1 is large enough, then maximum degree of every graph is at most $\frac{\Psi}{k+1} - 1$, w.h.p.*

Proof. We have $\Psi = \Delta + d\sqrt{k\Delta \log n}$ in Algorithm 1, therefore:

$$\frac{\Psi}{k+1} = \frac{\Delta}{k+1} + \frac{d\sqrt{k\Delta\log n}}{k+1} = \frac{\Delta}{k} + \Theta(\sqrt{\Delta \log n/k}),$$

where the hidden constants in the second term of the last equation can be made arbitrarily large depending on the choice of constant d. On the other hand, recall from Claim 4 that the maximum degree in any of the subgraphs is also at most $\frac{\Delta}{k} + O(\sqrt{\Delta \log n/k})$. Thus, the palette sizes are sufficient to color the subgraphs if d is a large enough constant. ◀

We are now ready to prove the algorithm's correctness.

▶ **Lemma 6.** *Algorithm 1 returns a proper edge coloring of G using $\Delta + O(\sqrt{k\Delta\log n})$ colors.*

Proof. The algorithm clearly uses $\Psi = \Delta + O(\sqrt{k\Delta\log n})$ colors, it remains to argue that the returned edge coloring is proper. Each subgraph (of type G_i or $G_{i,j}$) is sent to a single machine and edge-colored there using the palette that it is assigned to. Since by Observation 5, each palette has at least $\Delta' + 1$ colors for Δ' being the max degree in the subgraphs, there will be no conflicts in the colors associated to the edges within a partition. We only need to argue that two edges e and f sharing a vertex v that belong to two different subgraphs are not assigned the same color. Note that all subgraphs of type G_i are vertex disjoint and all receive the special color palette C', thus there cannot be any conflict there. To complete the proof, it suffices to prove that any two subgraphs $G_{i,j}$ and $G_{i',j'}$ that share a vertex receive different palettes. Note that in this case, either $i = i'$ or $j = j'$ by the partitioning. Assume w.l.o.g. that $i = i'$ and thus $j \neq j'$. Based on Algorithm 1 for $G_{i,j}$ and $G_{i',j'}$ to be assigned the same color palette, it should hold that

$$((i + j) \bmod k) + 1 = ((i' + j') \bmod k) + 1.$$

Since $i = i'$, this would imply that $(j \bmod k) = (j' \bmod k)$, though this would not be possible given that both j and j' are in $[k]$ and that $j \neq j'$. Therefore, any two subgraphs that share a vertex receive different palettes and thus there cannot be any conflicts, completing the proof. ◀

Next, we turn to prove the space bounds.

▶ **Lemma 7** (Implementation and Space Complexity). *Algorithm 1 can be implemented with total space $O(m)$ and space per machine of $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ w.h.p.*

Proof. We start with an implementation that uses the specified space per machine but can be wasteful in terms of the total space, then describe how we can overcome this problem and also achieve an optimal total space of $O(m)$.

We can use $k + \binom{k}{2}$ machines, each with a space of size $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ to assign colors to the edges in parallel. The first $m_1, \ldots, m_k$ machines will be used for edge coloring on $G_1, G_2, \ldots, G_k$ respectively. The other $m_{k+1}, \ldots, m_{k+\binom{k}{2}}$ machines will be used for edge coloring on the $G_{i,j}$ graphs. Lemma 4 already guarantees that each subgraph has size $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ w.h.p., and thus fits the memory of a single machine.

In the implementation discussed above, since the machines use $\widetilde{O}(n\Delta/k^2)$ space and there are $O(k^2)$ machines, the total memory can be $\widetilde{O}(n\Delta)$ which may be much larger than $O(m)$. This is because we allocate $O(n\Delta/k^2)$ space to each machine regardless of how much data it actually received. Though, observe that each edge of the graph belongs to exactly one of the subgraphs, i.e., the machines together only handle a total of $O(m)$ data. So we must consolidate into fewer machines. We do this by putting multiple subgraphs in each machine.

We start by recalling a sorting primitive in the MPC model which was proved in [26]. Basically, if there are N items to be sorted and the space per machine is $N^{\Omega(1)}$, then the algorithm of [26] sorts these items into the machines within $O(1)$ rounds. To use this primitive, we first label each edge $e = (u, v)$ of the graph by its subgraph name (e.g. G_i or $G_{i,j}$) which can be determined solely based on the end-points of the edge. After that, we sort the edges based on these labels. This way, all the edges inside each subgraph can be sent to the same machine within $O(1)$ rounds while also ensuring that the total required space remains $O(m)$. ◀

The algorithm for Theorem 1 was formalized as Algorithm 1. We showed in Lemma 6 that the algorithm correctly finds an edge coloring of the graph with the claimed number of colors. We also showed in Lemma 7 that the algorithm can be implemented with $O(m)$ total space and $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ space per machine. This completes the proof of Theorem 1.

3 Streaming Algorithms

We start in Section 3.1 by describing our streaming algorithm and its analysis when the arrival order is random. Then in Section 3.2, we give another algorithm for adversarial order streams.

3.1 Random Edge Arrival Setting

In this section, we give a streaming algorithm for $O(\Delta)$ edge coloring using $\widetilde{O}(n)$ space where the edges come in a random stream. That is, a permutation over the edges is chosen uniformly at random and then the edges arrive according to this permutation.

We first note that if $\Delta = O(\log n)$ then the problem is trivial as we can store the whole graph and then report a $\Delta + 1$ edge coloring (even without knowledge of Δ). As such, we assume $\Delta = \omega(\log n)$.

The algorithm — formalized as Algorithm 2 — maintains a counter c_v for each vertex v. At any point during the algorithm, this counter c_v basically denotes the highest color number used for the edges incident to v so far, plus 1. Therefore, upon arrival of an edge (u, v), it is safe to color this edge with $\max(c_u, c_v)$ as all edges incident to u and v have a color that is strictly smaller than this. Then, we increase the counters of both v and u to $\max(c_u, c_v) + 1$. It is not hard to see that the solution is always a valid coloring, in the remainder of this section, we mainly focus on the number of colors required by this algorithm and show that w.h.p., it is only $O(\Delta)$ for random arrivals.

Algorithm 2: Edge coloring for random streams.

```
Result: A feasible coloring $\mathcal{C} : E \to [\Psi]$ for a given graph $G = (V, E)$ with maximum
        degree $\Delta$ in a random stream
$c_v \leftarrow 0 \quad \forall v \in V$;
while $(u, v)$ is read from stream do
    $\mathcal{C}(u, v) \leftarrow \max(c_u, c_v)$;
    $c_u, c_v \leftarrow \mathcal{C}(u, v) + 1$;
end
```

We start by noting that this algorithm can actually be extremely bad if the order is adversarial. To see this, consider a path of size n. In an adversarial stream where the edges arrive in the order of the path, Algorithm 2 uses as many as $n - 1$ colors while the maximum degree is only 2! It is easy to see why this example is very unlikely to occur in random order streams: For a fixed path, it is very unlikely that the edges are randomly ordered in this very specific way.

To make this intuition rigorous for general graphs, we first prove the following crucial lemma which gives us the correct parameter to bound.

▶ **Lemma 8.** *Let Ψ be the size of the longest monotone (in the order of arrival) path in the line-graph of G. Then Algorithm 2 uses exactly Ψ colors.*

Proof. Take a monotone path $v_1, v_2, \ldots, v_\Psi$ in the line-graph of G and let $e_1, e_2, \ldots, e_\Psi$ be the edges of the original graph that correspond to these vertices respectively, i.e., e_1 arrives before e_2 which arrives before e_3 and so on. Since for any i, v_i and v_{i+1} are neighbors in the line-graph, then e_i and e_{i+1} should share an end-point v. This means that at the time of arrival of e_{i+1}, we have $c_v \geq \mathcal{C}(e_i) + 1$ which in turn, implies $\mathcal{C}(e_\Psi) > \mathcal{C}(e_{\Psi-1}) > \ldots > \mathcal{C}(e_1)$. Therefore, $\mathcal{C}(e_\Psi) \geq \Psi$.

On the other hand, suppose that there is an edge $e_1 = (u, v)$ for which $\mathcal{C}(e_1) = \Psi$ in Algorithm 2. This means that at least one of c_u or c_v equals Ψ when e_1 arrives, say c_u w.l.o.g. Let e_2 be the last edge incident to u that has arrived before e_1. It should hold that $\mathcal{C}(e_2) = \Psi - 1$. Using the same argument, for each $1 < i \leq \Psi$, we can find a neighboring edge e_i such that $\mathcal{C}(e_i) = \mathcal{C}(e_{i-1}) - 1$. This way, we end up with a sequence $e_1, \ldots, e_\Psi$ of edges, the path corresponding to this sequence in the line graph will be a monotone path of length Ψ, completing the proof. ◀

▶ **Theorem 9.** *There is a streaming edge coloring algorithm that for any graph $G = (V, E)$ uses at most $(2e + \epsilon)\Delta \approx 5.44\Delta$ colors w.h.p. for any constant $\epsilon > 0$ given that the edges in E arrive in a random order.*

Proof. We first prove that Algorithm 2 gives us a feasible coloring of graph G. Consider two edges $e_1 = (u, v)$ and $e_2 = (u, v')$ incident to vertex u such that e_1 appears earlier than e_2 in the stream. For any edge e we represent by $\mathcal{C}(e)$ the color assigned to that by the algorithm. After the algorithm colors e_1 with $\mathcal{C}(e_1)$, it sets c_u to $\mathcal{C}(e_1) + 1$. Thus, c_u is at least $\mathcal{C}(e_1) + 1$

when e_2 arrives and $\mathcal{C}(e_2) \geq \mathcal{C}(e_1) + 1$ consequently. Therefore, $\mathcal{C}(e_2) > \mathcal{C}(e_1)$ for any pair of edges incident to a common vertex, and $\mathcal{C}$ is a feasible coloring.

Next, for some constant α that we fix later, we show that the probability that an edge is assigned a color number at least $\alpha\Delta$ is at most n^{-c} for some constant $c \geq 2$, implying via a union bound over all the edges that indeed w.h.p., $\Psi \leq \alpha\Delta$.

We showed in Lemma 8 that if the number of colors Ψ used is $\alpha\Delta$, then there should exist a monotone path in the line-graph with size at least $\alpha\Delta$. Let $e_0, e_2, \ldots, e_{\alpha\Delta}$ be the corresponding edges to this path. Thus, it suffices to bound the probability of this event. Let Π denote the set of all such paths in the line graph. For a specific path $\pi \in \Pi$, the probability that it is monotone is $1/(\alpha\Delta)!$. Call this event X_π. On the other hand, we can upper bound the number of such paths by $(2\Delta)^{\alpha\Delta}$, i.e., $|\Pi| \leq (2\Delta)^{\alpha\Delta}$. This follows from the fact that each path should start from the corresponding vertex to e_0 in the line-graph, and that maximum degree in the line graph is $2\Delta - 2$ (which is the upper bound on the number of neighboring edges to each edge). Thus:

$$\Pr[\mathcal{C}(e_0) \geq \alpha\Delta] = \Pr[\bigvee_{\pi\in\Pi} X_\pi] \leq \sum_{\pi\in\Pi} \Pr[X_\pi = 1] \leq \frac{(2\Delta)^{\alpha\Delta}}{(\alpha\Delta)!},$$

where the last inequality is obtained by replacing $\Pr[X_\pi = 1]$ and $|\Pi|$ by the aforementioned bounds. Taking the logarithm of each side of the inequality, we get

$$\begin{aligned} \ln(\Pr[\mathcal{C}(e_0) \geq \alpha\Delta]) &\leq \alpha\Delta\ln(2\Delta) - \ln((\alpha\Delta)!) \\ &\leq \alpha\Delta\ln(2\Delta) - ((\alpha\Delta + 1/2)\ln(\alpha\Delta) - \alpha\Delta) \qquad (1) \\ &= \alpha\Delta\ln(2e/\alpha) - 1/2\ln(\alpha\Delta) \qquad (2) \\ &\leq \alpha\Delta\ln(2e/\alpha). \qquad (3) \end{aligned}$$

To obtain (1), we use Stirling's approximation of factorials to lower-bound $\ln((\alpha\Delta)!)$. Finally, we rearranged terms to imply (2). By plugging in $\alpha = 2e(1+\epsilon)$, we get

$$\begin{aligned} \ln(\Pr[\mathcal{C}(e_0) \geq 2e(1+\epsilon)\Delta]) &\leq 2e(1+\epsilon)\Delta\ln\left(\frac{1}{1+\epsilon}\right) \\ &= -2e(1+\epsilon)\ln(1+\epsilon)\Delta \\ &\leq -2e(1+\epsilon)\ln(1+\epsilon)\frac{c}{2e(1+\epsilon)\ln(1+\epsilon)}\ln(n) \qquad (4) \\ &= -c\ln(n) \end{aligned}$$

Since $\Delta = \omega(\log(n))$, we have $\Delta > c'\ln(n)$ for any constant c'. Inequality (4) follows from setting $c' = c/(2e(1+\epsilon)\ln(1+\epsilon))$ in $\Delta > c'\ln(n)$, where c is the constant for which we want to show the probability is upper-bounded by n^{-c}. Hence,

$$\Pr[\mathcal{C}(e_0) \geq 2e(1+\epsilon)\Delta] \leq n^{-c}.$$

Thus, Algorithm 2 returns a feasible coloring of the input graph G using at most $2e(1+\epsilon)\Delta$ colors, for any constant $\epsilon > 0$ w.h.p. if the edges arrive in a random order. ◀

To further evaluate the performance of Algorithm 2, we implemented and ran it for cliques of different size. The result of this experiment is provided in Table 1. The numbers are obtained by running the experiment 100 times and taking the average number of colors used. As it can be observed from Table 1, for cliques of size 100 to 1000, the number of colors used by the algorithm is in range $[3.3\Delta, 3.9\Delta]$ and it slightly increases by the size of the graph. Our analysis, however, shows that it should never exceed 5.44Δ.

Clique Size	100	200	300	400	500	600	700	800	900	1000
Colors Used	3.363Δ	3.563Δ	3.665Δ	3.717Δ	3.756Δ	3.787Δ	3.815Δ	3.838Δ	3.849Δ	3.863Δ

Table 1 The number of colors used by Algorithm 2 on cliques averaged over 100 trials.

3.2 Adversarial Edge Arrival Setting

In this section, we turn to arbitrary (i.e., adversarial) arrivals of the edges. We assume that the adversary is *oblivious*, i.e., the order of the edges is determined before the algorithm starts to operate so that the adversary cannot abuse the random bits used by the algorithm. Having this assumption, we give a randomized algorithm that w.h.p., outputs a valid edge coloring of the graph using $O(\Delta^2)$ colors while using $\tilde{O}(n)$ space. The algorithm is formalized as Algorithm 3. We note that this algorithm, as stated, requires knowledge of Δ. However we later show that we can get rid of this assumption. Overall, we get the following result:

Algorithm 3: Edge coloring in the adversarial order

```
Result: A feasible coloring for a given graph $G = (V, E)$ with maximum degree $\Delta$
for any vertex $v \in V$ do
    $r_v \leftarrow$ a sequence of $\log(n)$ independent random bits.
    for any $i \in [\log n]$ do
        $c_{v,i} \leftarrow 0$
    end
end
for any edge $e = (u, v)$ in the stream do
    Let $i$ be the smallest index for which $r_{v,i} \neq r_{u,i}$.
    if $\Delta 2^{-i} > \log n$ then
        if $r_{u,i} = 1$ then
            Assign color $(c_{u,i}, c_{v,i}, i)$ to $e$.
        else
            Assign color $(c_{v,i}, c_{u,i}, i)$ to $e$.
        end
        Increase both $c_{v,i}$ and $c_{u,i}$ by one.
    else
        Store edge $e$.
    end
end
Color the stored edges using a new set of colors.
```

▶ **Theorem 10.** *Given a graph G with maximum degree Δ, there exists a one pass streaming algorithm, that outputs a valid edge coloring of the G using $O(\Delta^2)$ colors w.h.p., using $\widetilde{O}(n)$ memory.*

Consider two vertices v and u and their string of random bits r_v and r_u defined in the algorihtm. Let $d_{u,v}$ be the smallest index i where $r_{u,i} \neq r_{v,i}$. Upon arrival of an edge $e = (u, v)$, we first find $i := d_{u,v}$. If $\Delta 2^{-i} > \log n$, we color the edge immediately. Otherwise, we store it. We will show that all the stored edges fit in the memory thus after reading all the stream we can color them with a palette of at most $\Delta + 1$ new colors. In the algorithm, for any vertex v and any $i \in [\log n]$, we define a counter $c_{u,i}$. If $\Delta 2^{-i} > \log n$ for any edge e, then we immediately assign e a color which is represented by a tuple $(c_{u,i}, c_{v,i}, i)$. Then, we increase counters $c_{u,i}$ and $c_{v,i}$. Note that we say two colors are the same if all three elements of them are equal. We first show that this gives us a valid coloring, which means it does not assign the same color to two edges adjacent to the same vertex. We use proof by contradiction. Assume that our algorithm assigns the same color to edges $e_1 = (u, v_1)$ and $e_2 = (u, v_2)$ adjacent to vertex u. None of them can be from the stored edges since we color them using a new palette. This means that $d_{u,v_1} = d_{u,v_2}$. Let us denote it by i. Without loss of generality, we assume that $r_{u,i} = 1$ and that in the input stream e_1 arrives before e_2. Note that the first element of the colors (which are tuples) assigned to these edges is the value of counter $c_{u,i}$ when they arrive. However, the algorithm increases $c_{u,i}$ by one after arrival of e_1 thus the colors assigned to e_1 and e_2 cannot be the same.

Now, it suffices to show that the total number of colors used by the algorithm is $O(\Delta^2)$. Given a vertex v, and a number $l \in [\log n]$ let us compute an upper-bound for counter $c_{v,i}$. Let N_v be the set of neighbors of this vertex and let $N_{v,i}$ be the set of neighbors like u where $d_{v,u} = i$. We know that $c_{v,i} = |N_{v,i}|$, thus given any vertex v and $i \in [\log(n)]$, we need to find a bound for $|N_{v,i}|$. Given any edge $e = (v, u)$ the probability of e being in set $N_{v,i}$ is 2^{-i} which means $\mathbb{E}[|N_{v,i}|] = \deg(v)2^{-i}$ where $\deg(v)$ is the degree of vertex v in the input graph.

Using a simple application of the Chernoff bound, for any vertex v, we get:

$$Pr\Big[|N_{v,i}| \geq \deg(v)2^{-i} + O\big(\sqrt{\deg(v)2^{-i}\log n}\big)\Big] \leq \frac{1}{n^c}.$$

Setting c to be a large enough constant, one can use union bound and show that w.h.p., for any vertex v and $i \in [\log n]$ where $\deg(v)2^{-i} \geq \log n$, we have $|N_{v,i}| \leq O(\deg(v)2^{-i})$.

Having this, we conclude that for any $i \in [\log n]$, where $\Delta 2^{-i} > \log n$, the number of colors used by the algorithm whose third element is i is at most $O(\Delta^2 2^{-2i})$ since the first and the second element of the color can get at most $O(\Delta 2^{-i})$ different values. Therefore, the total number of colors used for any such i is at most $O\big(\sum_{i\in[\log n]} \Delta^2 2^{-2i}\big) = O(\Delta^2)$. We will also show that the stored edges fit in the memory and thus we can color them using $O(\Delta)$ new colors. As a result the total number of colors used is $O(\Delta^2)$.

To give an upper-bound for the number of stored edges we first show that the expected number of stored edges for each vertex is $O(\log n)$. Let $j := \log(\frac{\Delta}{\log n})$. Recall that we store an edge (u, v) when $\Delta 2^{-d_{u,v}} < \log n$. Thus the expected number of stored edges adjacent to a single vertex v is at most

$$\sum_{j\leq i\leq \log n} d_v 2^{-i} \leq \sum_{j\leq i\leq \log n} \Delta 2^{-i} \leq \sum_{j\leq i\leq \log n} \log(n)2^{-i+j} = O(\log n).$$

To get the last equation we use the fact that $\Delta 2^{-j} \leq \log n$. By a similar argument that we used above (using Chernoff and Union bounds), with a high probability the total number of stored edges is $O(n \log n)$ which can be stored in the memory. Therefore the proof of this theorem is completed.

Knwoledge of Δ. As written, our algorithm depends on the knowledge of Δ because we must check $\Delta 2^{-i} > \log n$. We can get rid of this condition by keeping track of the degree $\deg_v^H$ of a vertex in the subgraph H we have seen so far, and then computing the max degree $\deg_{max}^H$. This only requires an additional $O(n)$ space. Thereafter, instead of checking if $\Delta 2^{-i} > \log n$, we check if $\deg_{max}^H 2^{-i} > \log n$. Whenever $\deg_{max}^H$ increases, we iterate over all stored edges and recompute whether or not $\deg_{max}^H 2^{-i} > \log n$. If so, we color the edge and remove it from the buffer, else we keep it. It is easy to see that this will not exceed the space bounds because at any timestep, we can assume the input graph was H in the first place. Then its max degree is $\Delta_H = \deg^H_{max}$, and we can apply the same argument for the space bounds as before, but using Δ_H instead of Δ. All other parts of the proof still hold. Therefore our algorithm does not require knowledge of Δ.

Finally, we remark that if one allows more space, then one can modify Algorithm 3 to use fewer number of colors. Though we focused only on the $\widetilde{O}(n)$ memory regime.

4 Open Problems

We believe the most notable future direction is to improve the number of colors used in our streaming algorithms. Specifically, our streaming algorithm for adversarial arrivals requires $O(\Delta^2)$ colors. A major open question is whether this can be improved to $O(\Delta)$ while also keeping the memory near-linear in n. Also for random arrival streams, we showed that Algorithm 2 achieves a 5.44Δ coloring and showed, experimentally, that it uses at least 3.86Δ colors. A particularly interesting open question is whether there is an algorithm that uses arbitrarily close to 2Δ colors using $\widetilde{O}(n)$ space in random arrival streams.

5 Acknowledgements

We thank the anonymous reviewers for many helpful comments and suggestions.

References

1 Kook Jin Ahn and Sudipto Guha. Access to Data and Number of Iterations: Dual Primal Algorithms for Maximum Matching under Resource Constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 202–211, 2015. URL: `http://doi.acm.org/10.1145/2755573.2755586`, `doi:10.1145/2755573.2755586`.

2 Noga Alon, László Babai, and Alon Itai. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *J. Algorithms*, 7(4):567–583, December 1986. URL: `http://dx.doi.org/10.1016/0196-6774(86)90019-2`, `doi:10.1016/0196-6774(86)90019-2`.

3 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583, 2014. URL: `http://doi.acm.org/10.1145/2591796.2591805`, `doi:10.1145/2591796.2591805`.

4 Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685, 2018. URL: `https://doi.org/10.1109/FOCS.2018.00070`, `doi:10.1109/FOCS.2018.00070`.

5 Eshrat Arjomandi. An efficient algorithm for colouring the edges of a graph with $\Delta+1$ colours. *INFOR: Information Systems and Operational Research*, 20(2):82–101, 1982.

6 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *Proceedings of the 30th annual ACM-SIAM Symposium on Discrete Algorithms (SODA), to appear*, 2019.

7 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear Algorithms for $(\Delta+1)$ Vertex Coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786, 2019. URL: `https://doi.org/10.1137/1.9781611975482.48`, `doi:10.1137/1.9781611975482.48`.

8 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The Locality of Distributed Symmetry Breaking. *J. ACM*, 63(3):20:1–20:45, June 2016. URL: `http://doi.acm.org/10.1145/2903137`, `doi:10.1145/2903137`.

9 MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. Affinity Clustering: Hierarchical Clustering at Scale. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6867–6877, 2017. URL: `http://papers.nips.cc/paper/7262-affinity-clustering-hierarchical-clustering-at-scale`.

10 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *J. ACM*, 64(6):40:1–40:58, 2017. URL: `http://doi.acm.org/10.1145/3125644`, `doi:10.1145/3125644`.

11 Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Brief Announcement: Semi-MapReduce Meets Congested Clique. *CoRR*, abs/1802.10297, 2018. URL: `http://arxiv.org/abs/1802.10297`, `arXiv:1802.10297`.

12 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Richard M. Karp. Massively parallel symmetry breaking on sparse graphs: MIS and maximal matching. *CoRR*, abs/1807.06701, 2018. URL: `http://arxiv.org/abs/1807.06701`, `arXiv:1807.06701`.

13 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab Mirrokni. Near-optimal massively parallel graph connectivity. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, to appear*, 2019.

14 Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G. Harris. Exponentially faster massively parallel maximal matching. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, to appear*, 2019.

15 Sebastian Brandt, Manuela Fischer, and Jara Uitto. Matching and MIS for uniformly sparse graphs in the low-memory MPC model. *CoRR*, abs/1807.05374, 2018. URL: `http://arxiv.org/abs/1807.05374`, `arXiv:1807.05374`.

16 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The Complexity of $(\Delta + 1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. *CoRR*, abs/1808.08419, 2018.

17 Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Round Compression for Parallel Matching Algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 471–484, 2018. URL: `http://doi.acm.org/10.1145/3188745.3188764`, `doi:10.1145/3188745.3188764`.

18 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 714–723, 2006. URL: `http://dl.acm.org/citation.cfm?id=1109557.1109635`.

19 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 531–543, 2004. URL: `https://doi.org/10.1007/978-3-540-27836-8_46`, `doi:10.1007/978-3-540-27836-8\_46`.

20 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 625–634, 2016. URL: `https://doi.org/10.1109/FOCS.2016.73`, `doi:10.1109/FOCS.2016.73`.

21 H. N. Gabow, T. Nishizeki, O. Kariv, D. Leven, , and O. Terada. Algorithms for edgecoloring graphs. Technical report, Tohoku University, 1985.

22 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 129–138, 2018. URL: `http://doi.acm.org/10.1145/3212734.3212743`, `doi:10.1145/3212734.3212743`.

23 Christian Glazik, Jan Schiemann, and Anand Srivastav. Finding euler tours in one pass in the w-streaming model with o(n log(n)) RAM. *CoRR*, abs/1710.04091, 2017. URL: `http://arxiv.org/abs/1710.04091`, `arXiv:1710.04091`.

24 A. Goldberg, S. Plotkin, and G. Shannon. Parallel Symmetry-breaking in Sparse Graphs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 315–324, New York, NY, USA, 1987. ACM. URL: `http://doi.acm.org/10.1145/28395.28429`, `doi:10.1145/28395.28429`.

25 Andrew V. Goldberg and Serge A. Plotkin. Parallel $(\Delta + 1)$-coloring of Constant-degree Graphs. *Inf. Process. Lett.*, 25(4):241–245, June 1987. URL: `http://dx.doi.org/10.1016/0020-0190(87)90169-4`, `doi:10.1016/0020-0190(87)90169-4`.

26 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pages 374–383, 2011. URL: `https://doi.org/10.1007/978-3-642-25591-5_39`, `doi:10.1007/978-3-642-25591-5_39`.

27 David G Harris, Johannes Schneider, and Hsin-Hao Su. Distributed $(\Delta + 1)$-coloring in sublogarithmic rounds. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 465–478. ACM, 2016.

28 Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and Local Ratio Algorithms in the MapReduce Model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, pages 43–52, New York, NY, USA, 2018. ACM. URL: `http://doi.acm.org/10.1145/3210377.3210386`, `doi:10.1145/3210377.3210386`.

29 Öjvind Johansson. Simple distributed *Delta*+1-coloring of graphs. *Inf. Process. Lett.*, 70(5):229–232, 1999. URL: `https://doi.org/10.1016/S0020-0190(99)00064-2`, `doi:10.1016/S0020-0190(99)00064-2`.

30 Tomasz Jurdzinski and Krzysztof Nowicki. MST in $O(1)$ rounds of congested clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2620–2632, 2018. URL: `https://doi.org/10.1137/1.9781611975031.167`, `doi:10.1137/1.9781611975031.167`.

31 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948, 2010. URL: `https://doi.org/10.1137/1.9781611973075.76`, `doi:10.1137/1.9781611973075.76`.

32 Fabian Kuhn and Rogert Wattenhofer. On the Complexity of Distributed Graph Coloring. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 7–15, New York, NY, USA, 2006. ACM. URL: `http://doi.acm.org/10.1145/1146381.1146387`, `doi:10.1145/1146381.1146387`.

33 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94, 2011. URL: `https://doi.org/10.1145/1989493.1989505`, `doi:10.1145/1989493.1989505`.

34 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM J. Comput.*, 21(1):193–201, February 1992. URL: `http://dx.doi.org/10.1137/0221015`, `doi:10.1137/0221015`.

35 M Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 1–10, New York, NY, USA, 1985. ACM. URL: `http://doi.acm.org/10.1145/22145.22146`, `doi:10.1145/22145.22146`.

36 Alessandro Panconesi and Aravind Srinivasan. Improved Distributed Algorithms for Coloring and Network Decomposition Problems. In *Proceedings of the Twenty-fourth Annual ACM*

Symposium on Theory of Computing, STOC '92, pages 581–592, New York, NY, USA, 1992. ACM. URL: `http://doi.acm.org/10.1145/129712.129769`, `doi:10.1145/129712.129769`.

37 Merav Parter. $(\Delta + 1)$ Coloring in the Congested Clique Model. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 160:1–160:14, 2018. URL: `https://doi.org/10.4230/LIPIcs.ICALP.2018.160`, `doi:10.4230/LIPIcs.ICALP.2018.160`.

38 Merav Parter and Hsin-Hao Su. Randomized $(\Delta + 1)$-Coloring in $O(\log^* \Delta)$ Congested Clique Rounds. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, pages 39:1–39:18, 2018. URL: `https://doi.org/10.4230/LIPIcs.DISC.2018.39`, `doi:10.4230/LIPIcs.DISC.2018.39`.

39 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.